ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM015

CONAN FOR BUILDING C++ TANGO DEVICES AT SOLEIL

P. Madela†, Y. M. Abiven, G. Abeillé, X. Elattaoui, J. Pham, F. Potier
Synchrotron SOLEIL, Gif-sur-Yvette, France

Abstract

At SOLEIL, our Tango devices are mainly developed in C++, with around 450 projects for building libraries and device servers for our accelerators and beamlines. We have a software factory that has enabled us to achieve continuous integration of our developments using Maven, which manages project dependencies. However, Maven is uncommon for C++. In addition, it has limitations that hinder us from supporting future platforms and new programming standards, leading us to replace it with Conan. Conan is a dependency and package manager for C and C++ that works on all platforms and integrates with various build systems. Its features are designed to enable modern continuous integration workflows with C++ and are an ideal alternative to Maven for our C++ build system. This transition is essential for the upgrade of SOLEIL (SOLEIL II), as we continue to develop new devices and update existing systems. We are confident that Conan will improve our development process and benefit our users. This paper will provide an overview of the integration process and describe the progress of deploying the new build system. We will share our insights and lessons learned throughout the transition process.

CONTEXT

At SOLEIL [1], our software development process relies on a well-established framework that encompasses various tools and practices, as illustrated in Fig. 1. This framework is crucial for ensuring the reliable operation of our accelerators and beamlines. It is the result of many years of work aimed at automating the delivery process, as documented in previous papers [2, 3]. The key components of our existing setup are as follows.

Deployment Process

Our deployment process involves a combination of Continuous Integration and Continuous Delivery (CI/CD) practices. Continuous integration is employed to build software artifacts automatically, ensuring that code changes are regularly integrated and tested. Additionally, we have a semi-automatic continuous delivery system in place to create packages that can be easily deployed. However, it's worth noting that the deployment of these packages currently requires manual intervention during each technical shutdown period.

Software Factories

We maintain two separate software factories to accommodate the diverse needs of our projects. The first one is dedicated to C++ development and is responsible for building approximately 400 Tango [4] device servers along with their respective libraries. Table 1 shows the number of C++ projects by platform. The second factory, specializing in Java development, oversees approximately 30 Tango device servers, along with libraries and graphical user interfaces.

Table 1: Number of C++ Projects

Type	Linux	Windows	Both
Application	1	0	0
Libraries	29	6	8
Devices	330	53	23

Tools of Software Factories

To support these software factories and facilitate our development workflows, we employ a set of tools that includes Gitlab Soleil [5] for version control, Jenkins [6] for automation, and Maven [7] for dependency management.

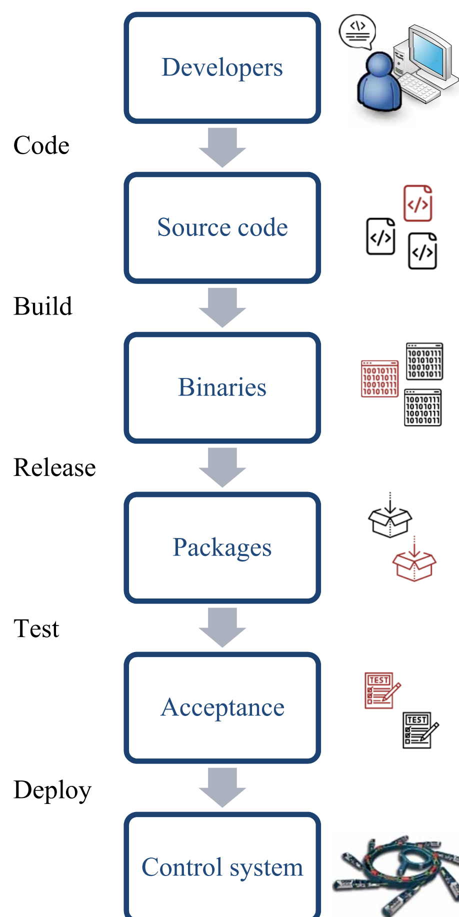

Figure 1: Software development process.

† patrick.madela@synchrotron-soleil.fr

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM015

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Limitations of Existing Infrastructure for C++

While our current system has proven effective, it does exhibit certain limitations, particularly in the context of C++ projects. Some of the noteworthy constraints include:

- **32-Bit Binary Limitation**: Our system is only capable of generating 32-bit binaries for Linux CentOS 6 [8] and Windows 7 [9]. It lacks support for crucial modern features, such as creating 64-bit programs and adhering to the latest C++ standards.
- **Outdated Components**: This limitation stems from the fact that certain components of our system are outdated and difficult to update. The process of modernizing this setup has become complex, further emphasizing the need for a more flexible and adaptable solution.
- **Maven Mismatch**: Another challenge lies in the mismatch between Maven and C++ development. While Maven has proven reliable for Java projects, it is not commonly used in the realm of C++. This mismatch makes it challenging for us to share our build approach with other institutes. Additionally, there is no built-in support within Maven for building third-party code, making it difficult to seamlessly integrate external libraries.

STRATEGY

Conan [10] as a Maven Alternative for C/C++

Recognizing the limitations of our current infrastructure for C++ projects, we have undertaken the task of identifying a suitable alternative. Our choice has led us to Conan, a dependency and package manager designed for C and C++ languages. Conan brings several distinct advantages:

- **Repository for C/C++**: Conan serves as a repository for managing packages and dependencies, offering a familiar structure akin to other package managers used in the software development world for other languages such as Maven, npm [11], and PyPI [12].
- **Abstract Build System**: Conan is an abstract build system, compatible with various other build systems like CMake [13], Make [14] and MSBuild [15]. It ensures seamless integration of packages from different development environments.
- **Multi-platform and Multi-binaries**: Conan's repository system excels in managing packages for multiple platforms and binaries, aligning well with our diverse software requirements.
- **Public Central Repository**: Conan provides easy access to a wide range of popular open-source C/C++ libraries through its central repository.
- **Continuous Integration Support**: Conan aligns well with our goal of establishing robust continuous integration workflows for C/C++ development.

Understanding Conan Recipes

In the context of Conan, a recipe serves as a fundamental element that guides the building and packaging of software components, encompassing applications, libraries, and tools. Essentially, a Conan recipe is a Python [16] script that encapsulates crucial information, enabling the construction and distribution of these components. The key components of a Conan recipe are as follow:

- **Specification of Dependencies**: A Conan recipe outlines the dependencies required for the successful construction of the software component. This includes specifying other libraries, tools, or packages upon which the component relies.
- **Build Instructions**: The recipe provides detailed instructions on how to build the component from source code. This encompasses compiler flags, build configurations, and any custom steps necessary to create the binary.
- **Metadata**: Metadata, such as version information, authorship, and licensing details are included within the recipe, ensuring transparency and clarity for both creators and consumers of the component.
- **Consumer Information**: A Conan recipe also offers essential details about the packaged library or tool, making it accessible and understandable to potential consumers.

To facilitate the construction and distribution of components within recipes, Conan provides a comprehensive set of tools. These tools simplify the management of dependencies, the generation of build files compatible with various build systems (e.g., CMake, Make, and MSBuild), and the integration of components into diverse development environments.

Our Forward-Looking Approach

We have chosen CMake as our primary build system, which is widely adopted and versatile for both Linux and Windows. Conan's recipes help us include dependencies and compilation options, ensuring efficient development and packaging of our Tango device servers.

In response to the challenges imposed by the obsolescence of components in our existing infrastructure, we have adopted a forward-looking strategy. This strategy involves the creation of a new infrastructure capable of accommodating both future platforms and legacy 32-bit environments.

REALIZATION

Transition to Conan

Our transition to Conan has followed these key steps:

1. **Learning Conan**: We started by learning how to use Conan.
2. **Building Necessary Libraries**: Next, we have built the libraries essential for the development of our Tango device servers.
3. **Proof Of Concept (POC)**: To validate the effectiveness of Conan and its compatibility with our infrastructure, we initiated several Proof of Concept (POC) projects. Two notable POCs includes the successful construction of the Lima [17] package, which couldn't be built on our existing infrastructure due to its 64-bit platform requirements. Additionally, we created a

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM015

Tango device for ChimeraTK [18], which is notably not natively buildable on our CentOS platforms.
4. **Automation**: Following these successful tests, we streamlined our workflow by automating certain aspects of the package-building process. This step was imperative to effectively manage the diversity of dependencies and platforms involved in our projects.
5. **Transition to Production**: The final step involves building and testing all our Tango device server projects using Conan and CMake instead of Maven. Additionally, we will conduct training sessions and facilitate knowledge transfer to support our developers in adapting to the changes.

Tools and Infrastructure

Throughout the various phases of our transition to Conan, significant effort was invested in establishing the necessary build tools and infrastructure. This included the development of up-to-date build tools for our legacy Linux and Windows environments, as well as the creation of Docker [19] images for Linux that integrated these tools for our build jobs.

We have also deployed the new software factory infrastructure, which incorporates the following principal components:

- Artifactory [20] as a Conan repository.
- Jenkins for automating build processes.
- Docker-based Jenkins agents for Linux.
- Jenkins agents for Windows with necessary build tools.

CONCLUSION

During our transition to Conan, we have identified both positive and negative aspects which can be summarized as following:

Pros of Conan

- **Package and Dependencies Management:** Conan provides a robust system for managing packages and their dependencies, simplifying the overall building process.
- **Multiplatform Support**: Its support for multiple platforms enhances flexibility, enabling us to address the diverse requirements of our projects on our legacy Linux and Windows platforms.
- **Flexibility with Build Systems**: Conan's compatibility with various build systems, including CMake and Make in our case, offers flexibility for integrating Tango with its dependencies and other components.
- **Version and Revision Control**: Conan offers robust version and revision control and can support both our current and future release processes.
- **Access to Popular Open-Source Libraries**: The public central repository facilitates easy access to a wide range of popular open-source C/C++ libraries.

Cons of Conan

- **Initial Learning Curve**: Conan requires an initial learning curve to become familiar with its features.
- **Introduces Additional Complexity**: The introduction of Conan may add complexity to the development process by adding an additional layer.
- **Still Requires Knowledge of Sub-Build System**: Despite its abstraction, working with Conan may still requires some understanding of underlying sub-build systems.
- **Requires Recent Version of Tools**: To fully leverage Conan's features, having up-to-date tool versions is necessary, which may pose challenges for our legacy systems.
- **Compatibility Limitation of Conan-Center Recipes**: Conan-Center recipes may have compatibility limitations that necessitates rewriting our own recipes for our legacy platform.

Conan as the Alternative for Our Build System

In conclusion, our adoption of Conan, integrated with the CMake build system, has proven to be a compelling alternative to replace our previous Maven-based build system. This transition has allowed us to simultaneously build devices for both legacy and future platforms, demonstrating our commitment to accommodating evolving technologies.

One notable advantage has been the simplified collaboration process while maintaining OS independence. Conan's centralized Jenkins pipeline templates and shared profiles/configurations between CI/CD and developer's environments will enhance our development workflow.

Furthermore, the shift to Conan has prepared us for upcoming challenges, including the migration to 64-bit and newer compilers/standards, the update of our deployment process, and the expansion of CI/CD capabilities for other domains, including those involving FPGA and firmware developments.

With Conan and CMake, we are well-equipped to develop future Tango device servers for the SOLEIL II [21] upgrade.

REFERENCES

[1] Synchrotron Soleil, `https://www.synchrotron-soleil.fr`

[2] A. Buteau *et al.*, "Making continuous integration a reality for control systems on a large scale basis", in *Proc., ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper TUP050, pp. 200-202.

[3] G. Abeillé *et al.*, "Continuous delivery at Soleil", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 51-55. `doi:10.18429/JACoW-ICALEPCS2015-MOD3O02`

[4] Tango Controls, `https://www.tango-controls.org`

[5] Conan, `https://conan.io`

[6] Gitlab Soleil, `https://gitlab.synchrotron-soleil.fr`

[7] Jenkins, `https://www.jenkins.io`

[8] Apache Maven, `https://maven.apache.org`

© Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THMBCM015

[9] CentOS, https://www.centos.org
[10] Microfoft Windows, https://windows.microsoft.com
[11] NPM, https://www.npmjs.com
[12] PyPI, https://pypi.org
[13] Cmake, https://cmake.org
[14] GNU Make, https://www.gnu.org/software/make
[15] Microsoft Visual Studio, https://learn.microsoft.com/en-us/visualstudio/msbuild
[16] Python, https://www.python.org
[17] LIMA, https://lima1.readthedocs.io/en/latest
[18] ChimeraTK, https://github.com/ChimeraTK
[19] Docker, https://www.docker.com
[20] JFrog Artifactory, https://jfrog.com/fr/artifactory
[21] Y. M. Abiven *et al.*, "SOLEIL II: Towards A Major Transformation of the Facility", presented at ICALEPCS'23, Cape Town, South Africa, Oct. 2023, paper TUMBCMO21, this conference.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI